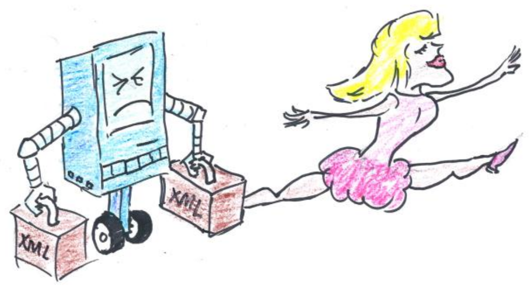
A Quick Introduction to *Open**EXI***

The *OpenEXI* Project

March, 2012

People Love XML

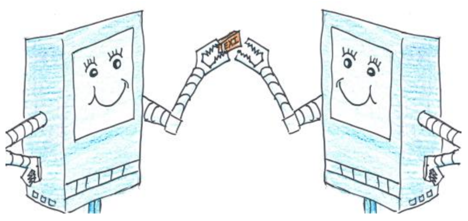
People love XML. The simple structure makes it easy to read. It's flexible, and has many applications. XML files are portable. They can be transferred from one platform to another without modification. All of this is great, for *people*.



But "portability" is a relative term. The truth is, to make XML easy for people to read, computers end up carrying a lot of unnecessary human baggage.

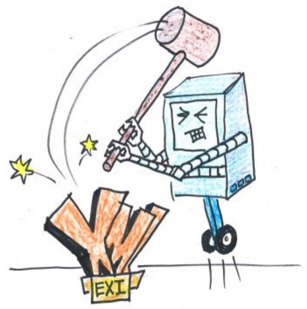
Computers don't need long labels in hierarchical format to give them visual context; computers can process a stream of information with no context at all.

We can reduce the burden on our computer co-workers by sending only the most essential data. When human-readable elements are replaced with concise encoding, files are substantially reduced in size. They can pass from one computer to the next at greater speed, process more quickly, and be restored with no loss of logical content.



EXI, the Efficient XML Interchange standard, is a W3C recommendation for enhancing the communication of XML data between computer systems.

EXI provides a more concise markup strategy for describing document structure. It describes a method for grouping information into *channels* that pack the most possible values into the smallest number of bits. These efficient files are then compressed using the Deflate standard to make them as small as possible. Smaller files transfer and process more quickly than standard XML files compressed with GZip or similar tools.

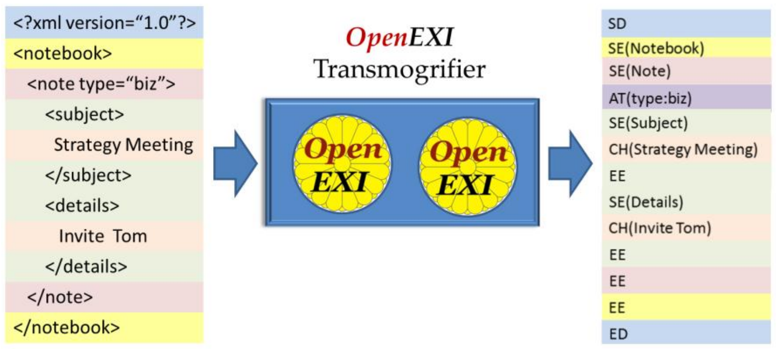


OpenEXI is an open source Java Application Programming Interface that provides utilities you can use to transfer eXtended Markup Language files to Efficient XML Interchange format and back again very quickly with no risk of data loss.

If you regularly transfer large XML files between machines, or you have devices with low memory or bandwidth, OpenEXI can substantially enhance your system's performance.

Default Processing

OpenEXI can convert any XML file to EXI format using the *Transmogrifier* class. If no schema information is available, the Transmogrifier constructs and records the structure of the file as it is converted.



As the SAX parser reads the file, each of the elements and attributes is handled as a separate *event*.

By default, non-essential information is not included in the transformed file. The initial processing instruction is replaced with a Start Document, or *SD*, marker. Even then, the SD marker is not stored in the final file. When decoded, OpenEXI assumes that a new file stream starts with an SD event, and ends with an ED event.

Every new element is replaced with an SE event. The first time it is used, the name string is stored with it. After that, the name is inferred. AT and CH events store the content data. EE events are never named: it can be assumed that each one closes the current nested SE event. Each of these markers is only 2-6 bits in size (despite their depiction here as 2 characters).

Removing the labels from the start and end elements in the file is just the starting point for reducing the file size, but it represents a significant savings in and of itself.

String Tables

Another reason that content can be stored so economically is EXI's use of a String Table. When *OpenEXI* encounters a unique string during processing, the string is added to the output stream, but also added to the String Table. The next time the string value appears in the file, *OpenEXI* marks a reference to the original value in the String Table rather than storing the string a second time. When there are many repeated values in a data set, this can mean a huge savings.

This XML file lists some favorite things. The favorite weapon is a *bow.* The bow string is added to the output stream and becomes the first entry in the String Table. A favorite decoration also happens to be a bow. This time, a reference to string 1 is added to the output stream. Every element that uses the characters b-o-w can use the reference rather than store the string.

XML

```
<?xml version="1.0"?>
<favorites>
  <weapon>
    bow
  </weapon>
  <decoration>
    bow
  </decoration>
  <gesture>
    bow
  </gesture>
  <stateroom>
    bow
  </stateroom>
  <leg>
    bow
  </leg>
</favorites>
```

String Table

String ID	String
S1	bow

EXI

```
SD
SE(favorites)
SE(weapon)
CH(bow)
EE
SE(decoration)
S1
EE
SE(gesture)
S1
EE
SE(stateroom)
S1
EE
SE(leg)
S1
EE
EE
ED
```

When processing is finished, the String Table is discarded.

On the receiving end, the String Table is reconstructed in the same order. The first occurrence of each string is added to the table as it is encountered. The values are always present before the tokens that reference them, and the original logical content of the file is restored.

If you are working with machines that have limited bandwidth or memory, you have the option of restricting the number of strings added to the table at one time (strings are replaced on a first-in, first-out basis). You can also limit the length of the strings included in the table, since longer strings are likely to be unique and will not provide any benefit from reuse.

Bit Alignment

There are three types of bit alignment used for EXI streams – byte-aligned, precompressed, and bit-packed. Each represents a further shift away from human-burdened XML to more efficient XML interchange files.

Byte-aligned Streams

Byte-aligned streams are the most human-readable productions from *OpenEXI*. These files are useful for troubleshooting. The information is encoded with normal byte boundaries intact, and you can still see some of the text in context. Token characters appear where the values are taken from the string table. You can see that, using the String Table, *OpenEXI* has already substantially reduced the size of the file.

Notebook.xml

```
<?xml version="1.0"?>
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Buy steaks.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
</notebook>
```

Notebook.exi

```
€       notebook note
        category
Business subject
        Strategy Meeting
     ??details
        Invite Tom.
     ??????????
        Invite Joan.
     ??
personal??
        Saturday BBQ
     ??????????????
        Buy steaks.
     ???????????
```

Byte-aligned files are not intended for transfer, because they have not been optimized. Looking at a byte-aligned version of a file can be helpful when you want to verify the results of your EXI transformation.

If a file has an associated XML schema, or *XSD*, the schema can speed processing by giving the parser information up front about the types of events in the file.

The names of defined element and attribute events are omitted from the EXI file.

When using default processing, any non-compliant events are encoded at the end of the file and integrated when the file is restored. The only requirement is that the XML is well-formed. The processing is more efficient thanks to the schema, and no data are lost.

If you are confident that your XML file is 100% compliant with the schema, you can use *strict interpretation* to encode the file. This is faster to process, and results in the smallest possible file size. When using strict interpretation, any variance from the schema (such as a string value where an integer is expected) throws an exception and the EXI compilation fails. This avoids production of an incomplete or corrupted file.

Here's the same file again, but this time it was converted to EXI with an XML schema definition. The structure information has been removed, and only the values are stored in the interchange file. For this use case, the receiver of the information should already have the schema on hand, and apply it when the file is restored to XML.

Notebook.xml

```
<?xml version="1.0"?>
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Buy steaks.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
</notebook>
```

Notebook.exi

```
€???
business??
           Strategy Meeting
       ???
           Invite Tom.
       ???????????
           Invite Joan.
       ????
personal??
           Saturday BBQ
       ????????????????
           Buy steaks.
       ??????????????
```

Precompressed Streams

Precompression alignment shows another way that EXI optimizes storage.

Human-readable XML has to maintain a hierarchical structure to make the data understandable to humans. Since the computer is not concerned with visual context, it can rearrange the data in the most efficient way for storage and transfer. By putting all of the like values together in *channels*, the information can be tightly packed, particularly with date, numeric, and Boolean values.

Looking at this default precompressed version of Notebook.xml, you can see that the structure information appears first. Next are the values for the attributes. All subject values are grouped together, and finally all of the detail values. When the file is decoded, the EXIReader takes a value from the top of each channel as needed to reconstruct the original file.

Notebook.xml

```
<?xml version="1.0"?>
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Buy steaks.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
</notebook>
```

Notebook.exi

```
€           notebook note
            category subject
??details?????????????????????????
??????
business?
personal??
    Strategy Meeting
  ?
    Saturday BBQ
  ??
    Invite Tom.

    Invite Joan.
  ??
    Buy steaks.
  ?
```

Compressed Streams

While not an alignment in and of itself, compression is orthogonal because the option dictates that EXI files be arranged in channels and not bit-packed.

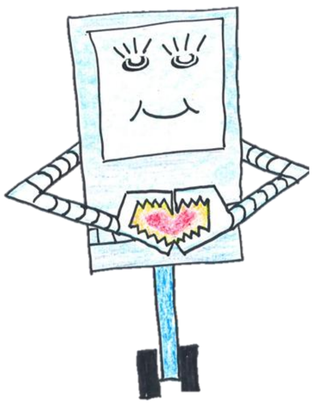
This is how the compressed information looks when opened in a text editor. While petty mortals cannot make heads or tails of it, computers can store and transfer EXI information efficiently, now that all human baggage has been removed.

Notebook.xml

```xml
<?xml version="1.0"?>
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Buy steaks.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
</notebook>
```

Notebook.exi

```
€c`ÀŒ\I¥Å™y©ÅÅ\©EÅùy‰9ŒŒr\`
\R”X’š^©à›šZ’™f‚É&”¥$V*89Bd
€Z%¡’žye™%©!ù¹z`)Tq¯üÄ<^L‹S
i¥BqIjbv1T†?
```

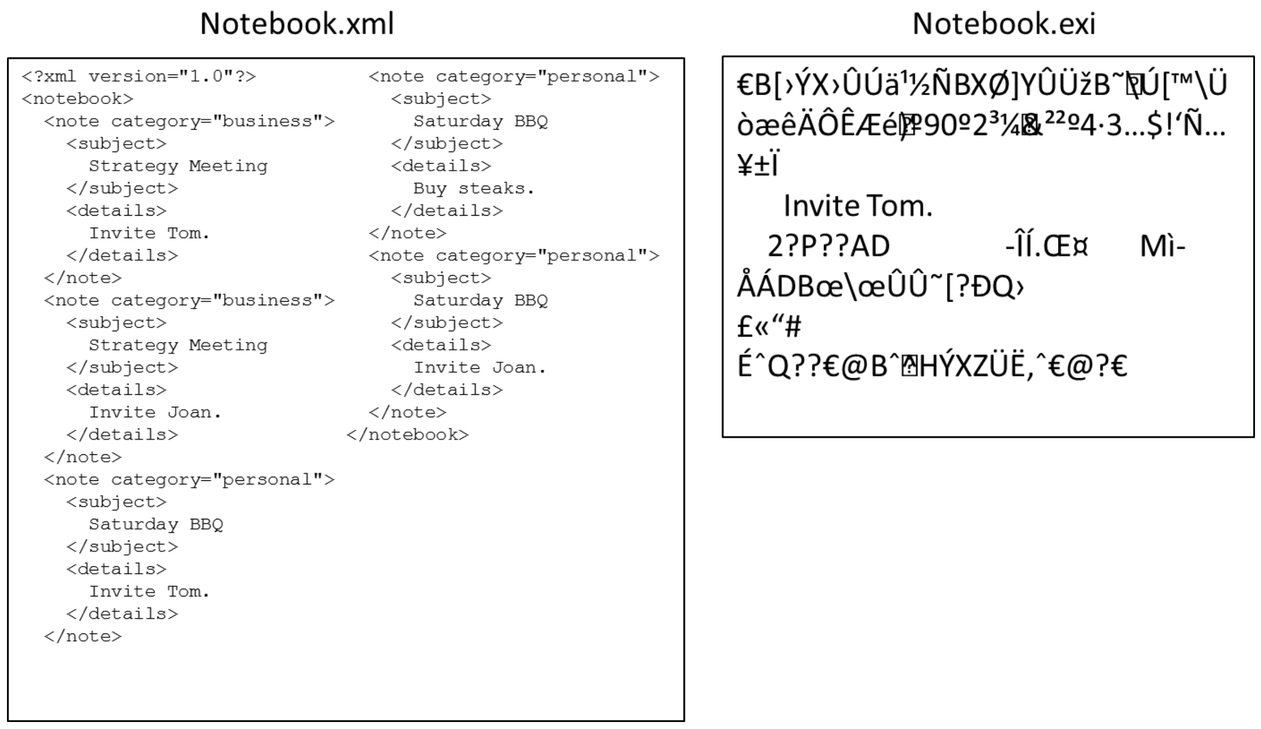


The use of channels, combined with the Deflate standard for reducing file size, results in a very compact representation of the data. Files process and transfer up to seven times faster with *Open*EXI versus using GZip to compress a standard XML file.

While optimal compression isn't necessary for all use cases, users who have very large data sets or who work with machines with limited storage or bandwidth can benefit greatly from compressing their data using *Open*EXI.

Bit-packed Streams

Bit-packed alignment is the default when transforming XML to EXI. The data are packed together in the stream without regard to byte boundaries. Sometimes this happens to result in readable text, if the boundaries between bytes incidentally align, but most of the file is unreadable to humans.


Notebook.xml
Notebook.exi

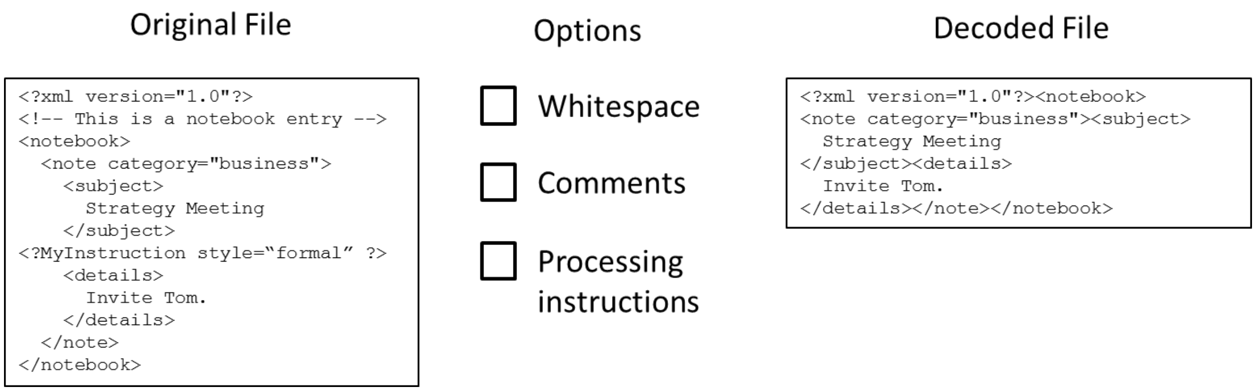
```
<?xml version="1.0"?>
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Tom.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Buy steaks.
    </details>
  </note>
  <note category="personal">
    <subject>
      Saturday BBQ
    </subject>
    <details>
      Invite Joan.
    </details>
  </note>
</notebook>
```

```
€B[›ÝX›ÛÚä¹½ÑBXØ]YÛÜžB˜ÚÚ[™\Ü
òæêÄÔÊÆé90º2³¼²²º4·3…$!'Ñ…
¥±Ï
    Invite Tom.
  2?P??AD           -ÎÍ.Œ¤      Mì-
ÅÁDBœ\œÛÛ˜[?ÐQ›
£«"#
ÉˆQ??€@B˜HÝXZÜË‚ˆ€@?€
```

There is a limit to how much benefit can be derived from file compression using the Deflate algorithm. For some smaller file sizes, greater compaction can be achieved using the default bit-packed setting.

Preservation Options

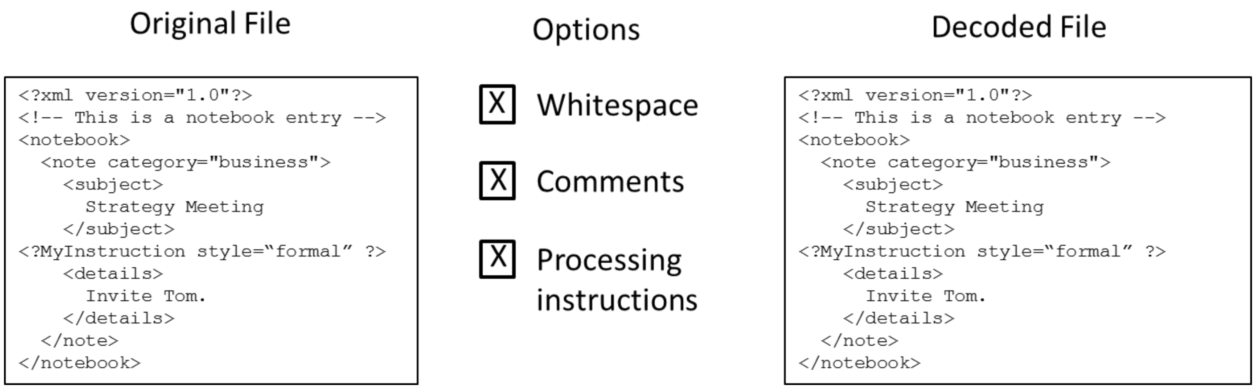
Whitespace, comments, and other non-essential content in XML files is there to benefit human users. If humans are not going to use the file after conversion, there is no reason to preserve this information. Omitting comments and whitespace can substantially reduce file size.



```
<?xml version="1.0"?>
<!-- This is a notebook entry -->
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
<?MyInstruction style=“formal” ?>
    <details>
      Invite Tom.
    </details>
  </note>
</notebook>
```

```
<?xml version="1.0"?><notebook>
<note category="business"><subject>
  Strategy Meeting
</subject><details>
  Invite Tom.
</details></note></notebook>
```

If your use case requires that you retain this information, the *OpenEXI* Transmogrifier gives you the option of saving any or all of these items.



```
<?xml version="1.0"?>
<!-- This is a notebook entry -->
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
<?MyInstruction style=“formal” ?>
    <details>
      Invite Tom.
    </details>
  </note>
</notebook>
```

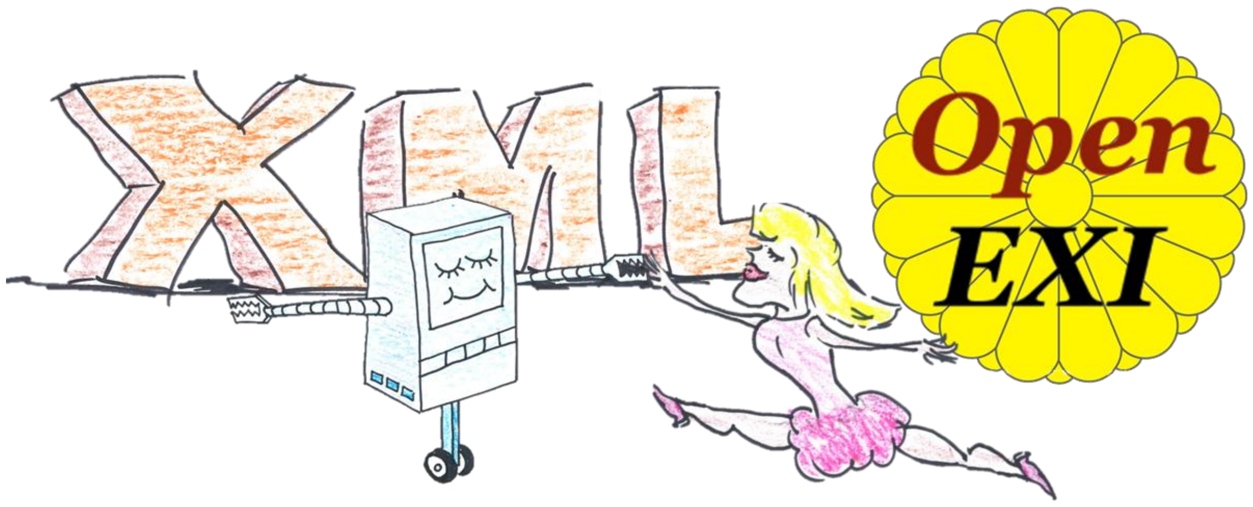
```
<?xml version="1.0"?>
<!-- This is a notebook entry -->
<notebook>
  <note category="business">
    <subject>
      Strategy Meeting
    </subject>
<?MyInstruction style=“formal” ?>
    <details>
      Invite Tom.
    </details>
  </note>
</notebook>
```

If you save only the information that you will use after transfer, you will gain the greatest increase in speed and reduction in file size.

The Best of Both Worlds

You can see that XML with *Open*EXI gives you the best of both worlds. XML information is convenient for people to work with, while EXI information is far more compact and transfers quickly. People get what they want without creating unnecessary work for the computer compatriots.

If you transfer large amounts of XML data, or your XML devices have low memory or bandwidth, OpenEXI could be the solution that meets your needs.



This whitepaper was prepared by Dennis Dawson with Takuki Kamiya of the *Open*EXI Project with support from Fujitsu Laboratories of America, March, 2012.